UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/767,512	01/29/2004	Adiel M. Yoaz	50277-2935	5970

42425 7590 06/03/2009

HICKMAN PALERMO TRUONG & BECKER/ORACLE
2055 GATEWAY PLACE
SUITE 550
SAN JOSE, CA 95110-1083

EXAMINER	
TRUONG, CAM Y T	
ART UNIT	PAPER NUMBER
2169	

MAIL DATE	DELIVERY MODE
06/03/2009	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/767,512	YOAZ ET AL.	
	Examiner	Art Unit	
	Cam Y T. Truong	2169	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *06 April 2009*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1,4,5,9,16,22,25-28,30 and 35-38* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1, 4-5, 9, 16, 22, 25-28, 30, 35-38* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. Applicant has amended claims 1, 4-5, 9, 16, 22, 25-28, 30, 35-38 in the amendment filed on 4/6/2009.

Claims 1, 4-5, 9, 16, 22, 25-28, 30, 35-38 are pending in this Office Action.

Response to Arguments

2. Applicant's arguments with respect to claims 1, 4-5, 9, 16, 22, 25-28, 30, 35-38 have been considered but are moot in view of the new ground(s) of rejection.

a. Applicant argued that claims 22, 26-28 and 30 are statutory and it is not clear to Applicant how punch cards or paper tape may be considered a form of energy as each are tangible objects (typically made of paper). In any case, Claims 22, 26-28, and 30 recite "a computer readable storage_ medium". By stating that each claim is a computer readable *storage medium,* each claim is directed towards a tangible object, and not to any form of energy. As such, Applicant respectfully requests reconsideration of the rejection of Claims 22, 26-28 and 30 under 35 U.S.C. § 101.

Examiner respectfully disagrees because in paragraph 0031, applicant has provided evidence that applicant intends the "medium" to include e.g, punch cards, paper tape. The punch cards and paper tape are not hardware medium. As such, the claim is drawn to a form of energy. Energy is not one of the four categories of invention and therefore this claim(s) is/are not statutory. Energy is not a series of steps or acts and thus is not a process. Energy is not a physical article or object and as such is not a

machine or manufacture. Energy is not a combination of substances and therefor not a composition of matter.

Thus, the 101 rejection for claims 22, 26-28 and 30 is maintained in this Office Action.

b. Applicant argued that the cited references does not teach the amended claim.

In response to applicant's argument, claims are rejected under new ground. In particularly, As to claim 1, Crisan teaches a method of operation within a data processing system (abstract), the method comprising:

"before receiving a request to execute a statement that requires computation of a first function, associating the first function with a second function" as before receiving a SQL statement the function generator generates an executable script file 250 that includes statements to register object oriented classes with DBMS 204 that enable the SQL program 202 to call UDFs 208a, 208b...208n to cause the invocation of the external workflow 220a, 220b, ...220n for which the UDF 208a, 208b…208n.

The UDF 208a, 208b . . . 208n statements are executed to set (at block 352) values within the object(s) to pass to the invoked workflow 220a, 220b . . . 220n that include parameters passed with the call from the SQL program 202. The executed statements within the UDF 208a, 208b . . . 208n would then invoke (at block 354) the workflow 220a, 220b . . . 220n by transmitting a workflow function call to the workflow engine 210 with the input object including values set from parameters received from the SQL program 202. Upon receiving (at block 356) a response from the workflow, the UDF 208a, 208b . . . 208n would

process (at block 358) the returned output to extract any value(s) returned from the workflow and return (at block 360) extracted value(s) or other information to the SQL program 202 that called the UDF 208a, 208b . . . 208n (paragraphs 0059-0060, 0144-0145), the UDF is represented as a first function, the workflow function is represented as a second function;

"that returns a data type descriptor for the first function" as workflow function return data type information not data type descriptor for the UDF (paragraphs 138-142);

"receiving a request to execute the statement that requires computation of the first function to return data from a source" as receiving a quest to execute a UDF to return data from a source (paragraph 0145);

"in response to receiving the request to execute the statement" (paragraphs 0058-0060), "performing the steps of: executing the second function to obtain the data type descriptor that identifies one or more data types of result data that should be returned for the first function" as the UDF 208a, 208b . . . 208n statements are executed to set (at block 352) values within the object(s) to pass to the invoked workflow 220a, 220b . . . 220n that include parameters passed with the call from the SQL program 202. The executed statements within the UDF 208a, 208b . . . 208n would then invoke (at block 354) the workflow 220a, 220b . . . 220n by transmitting a workflow function call to the workflow engine 210 with the input object including values set from parameters received from the SQL program 202. Upon receiving (at block 356) a response from the workflow, the UDF 208a, 208b . . . 208n would

process (at block 358) the returned output to extract any value(s) returned from the workflow and return (at block 360) extracted value(s) or other information to the SQL program 202 that called the UDF 208a, 208b . . . 208n (paragraph 0145);

For instance, if the return data is of type "scalar" as indicated in the function field 292, then a single value is returned. If the type is "table", then a table is returned, and the function generator 244 will have to generate code in the UDF source file 248 to get the number of rows to return as indicated in the rows field 296. Code is further generated (at block 322) into the UDF file 248 to return any extracted value/values to the calling SQL program 202. After generating all the necessary code in the UDF file 248 to set values in an input object, invoke the workflow with the set input object, and extract any return values to return to the calling SQL program 202, the UDF source file 248 is returned (at block 324). The function generator 244 further generates an executable script file 250 that includes statements to register object oriented classes with the DBMS 204 that enable the SQL program 202 to call the UDFs 208a, 208b . . . 208n (paragraph 0144). The above information shows that a second function returns data type information but not data type descriptor;

“registering the one or more data types with a data processing system” as registering data types with a data processing system (paragraph 0129);

“compiling the statement based on the registered one or more data types” as (paragraphs 134, 136, 0052);

"executing the first function to obtain the result data" executing the UDF to obtain the result data (paragraphs 0145, 0117-0118);

"storing the result data obtained from the source" as storing the result data obtained from the resource (paragraphs 0058, 0120);

"returning the result data as data in the format of the registered the one or more data types" as the UDF 208a, 208b . . . 208n statements are executed to set (at block 352) values within the object(s) to pass to the invoked workflow 220a, 220b . . . 220n that include parameters passed with the call from the SQL program 202. The executed statements within the UDF 208a, 208b . . . 208n would then invoke (at block 354) the workflow 220a, 220b . . . 220n by transmitting a workflow function call to the workflow engine 210 with the input object including values set from parameters received from the SQL program 202. Upon receiving (at block 356) a response from the workflow, the UDF 208a, 208b . . . 208n would process (at block 358) the returned output to extract any value(s) returned from the workflow and return (at block 360) extracted value(s) or other information to the SQL program 202 that called the UDF 208a, 208b . . . 208n (paragraph 0145);

For example, the UDF may then be invoked within standard SQL statements such as the following: values getRate(`UK`,`USA`) select item, (price * getRate(country, `USA`)) as cost from purchase_orders (paragraph 0117);
In addition to creating scalar functions such as above, one may also create table functions where the invocation of the web service yields a table of one or more rows,

as represented by the backward arrow in step b) of FIG. 9. For instance, assume that there is a web service that returns status about commercial flights. Using the UDF generating tool one may generate a table function getFlightInfo that returns a table containing the airline, flight number, location, speed, altitude and equipment (paragraph 0118);

"wherein the method is performed by one or more computing device" as (abstract, paragraph 0031).

Crisan does not explicitly teach the claimed limitation "data type descriptor; in a format that reflects registered one or more data types".

Muthy teaches when an XML document is stored in a database server that supports the XML schema registration techniques described herein, the database server is able to validate the documents to verify that they confirm to the corresponding XML schema. The validation may include validation of both the structure and the datatypes used by the XML document (paragraph 0215).

Murthy teaches creation of SQL Object Types when an XML schema is registered, database server 104 creates the appropriate SQL object types that enable a structured storage of XML documents conforming to this schema. All SQL object types are created in the current user's schema (by default). For example, when PO.xsd is registered, the following SQL types are created. Create type Item_t as object (part varchar2(1000), price number); create type Item_varray_t as varray(1000) of OBJ_T1; create type PurchaseOrder_t as object (purchasedate

date, ponum number, company varchar2(100), item Item_varray_t) (paragraph [0100]).

For example, if the XML schema specifies the datatype to be "string" and a maxLength value of less than 4000, it gets mapped to a varchar2 attribute of the specified length. However, if the maxLength value is not specified in the XML schema, it can only be mapped to a LOB. This is sub-optimal in cases when the majority of string values are actually small--and a very small fraction of them is large enough to necessitate a LOB. The ideal SQL datatype would be varchar2(*) that would perform like varchars for small strings but can accommodate larger strings as well. Further, such columns should support all varchar functionality such as indexing, SQL functions, etc. A similar case can be made for needing a raw(*) datatype to hold unbounded binary values without loss of performance and/or functionality for the small cases (paragraph 0123).

The user can specify SQLType to be NVARCHAR2 for a particular string element or attribute. This ensures that NVARCHAR2 is chosen as the SQL type for the particular element/attribute (paragraph 0126).

Assuming that the XML schema identified by "http://www.oracle.com/P-O.xsd" has already been registered. A XMLType table can be created to store instances conforming to the PurchaseOrder element of this schema--in an object-relational format--as follows: create table MyPOs of xmltype; element "http://www.oracle.com/PO.xsd#PurchaseOder".

Hidden columns are created corresponding to the object type to which the

PurchaseOrder element has been mapped. In addition, a XMLExtra object column is created to store the top-level instance data such as namespaces declarations, etc. Note: XMLDATA is a pseudo-attribute of XMLType that allows directly accessing the underlying object column (abstract, paragraphs 0128-0130).

The above information shows that XML documents are stored based on type information.

Anderson teaches a function returns a char variable as type descriptor containing the full pathname of BLOBs that are stored in external servers to identify a BLOB as data type (col. 26, lines 35-67).

It would have been obvious to a person of an ordinary skill in the art at the time the invention was made to apply Murthy's teaching of storing XML documents based on type information and Anderson's teaching of a function returns a char variable containing the full pathname of BLOBs that are stored in external servers to identify a BLOB as data type to Crisan's system in order to determine how to store subsequently result XML data that conform to the registered XML schema so that the performance and scalability features of the database can be fully exploited to access the result XML data.

Claim Rejections - 35 USC § 101

3. 35 U.S.C. 101 reads as follows:

Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

4. Claims 22, 26-28, 30 are rejected under 35 U.S.C. 101 because the claims fail to place the invention squarely within one statutory class of invention. On paragraph 0031 of the application, applicant has provided evidence that applicant intends the "medium" to include punch card, paper tape. The punch cards and paper tape are not hardware media. As such, the claim is drawn to a form of energy. Energy is not one of the four categories of invention and therefore this claim(s) is/are not statutory. Energy is not a series of steps or acts and thus is not a process. Energy is not a physical article or object and as such is not a machine or manufacture. Energy is not a combination of substances and therefor not a composition of matter.

Claim Rejections - 35 USC § 103

5. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

6. Claims 1, 4-5, 16, 22, 25-28 and 35-37 are rejected under 35 U.S.C. 103(a) as being unpatentable over Crisan et al (or hereinafter "Crisan") (US 20030191769) in view of

Murthy et al (or hereinafter "Murthy") (US 20030140308) and Anderson et al (or hereinafter "Anderson") (US 6047291).

As to claim 1, Crisan teaches a method of operation within a data processing system (abstract), the method comprising:

"before receiving a request to execute a statement that requires computation of a first function, associating the first function with a second function" as before receiving a SQL statement the function generator generates an executable script file 250 that includes statements to register object oriented classes with DBMS 204 that enable the SQL program 202 to call UDFs 208a, 208b...208n to cause the invocation of the external workflow 220a, 220b, ...220n for which the UDF 208a, 208b…208n.

The UDF 208a, 208b . . . 208n statements are executed to set (at block 352) values within the object(s) to pass to the invoked workflow 220a, 220b . . . 220n that include parameters passed with the call from the SQL program 202. The executed statements within the UDF 208a, 208b . . . 208n would then invoke (at block 354) the workflow 220a, 220b . . . 220n by transmitting a workflow function call to the workflow engine 210 with the input object including values set from parameters received from the SQL program 202. Upon receiving (at block 356) a response from the workflow, the UDF 208a, 208b . . . 208n would process (at block 358) the returned output to extract any value(s) returned from the workflow and return (at block 360) extracted value(s) or other

information to the SQL program 202 that called the UDF 208a, 208b . . . 208n (paragraphs 0059-0060, 0144-0145), the UDF is represented as a first function, the workflow function is represented as a second function;

"that returns a data type descriptor for the first function" as workflow function return data type information not data type descriptor for the UDF (paragraphs 138-142);

"receiving a request to execute the statement that requires computation of the first function to return data from a source" as receiving a quest to execute a UDF to return data from a source (paragraph 0145);

"in response to receiving the request to execute the statement" (paragraphs 0058-0060), "performing the steps of: executing the second function to obtain the data type descriptor that identifies one or more data types of result data that should be returned for the first function" as the UDF 208a, 208b . . . 208n statements are executed to set (at block 352) values within the object(s) to pass to the invoked workflow 220a, 220b . . . 220n that include parameters passed with the call from the SQL program 202. The executed statements within the UDF 208a, 208b . . . 208n would then invoke (at block 354) the workflow 220a, 220b . . . 220n by transmitting a workflow function call to the workflow engine 210 with the input object including values set from parameters received from the SQL program 202. Upon receiving (at block 356) a response from the workflow, the UDF 208a, 208b . . . 208n would

process (at block 358) the returned output to extract any value(s) returned from the workflow and return (at block 360) extracted value(s) or other information to the SQL program 202 that called the UDF 208a, 208b . . . 208n (paragraph 0145);

For instance, if the return data is of type "scalar" as indicated in the function field 292, then a single value is returned. If the type is "table", then a table is returned, and the function generator 244 will have to generate code in the UDF source file 248 to get the number of rows to return as indicated in the rows field 296. Code is further generated (at block 322) into the UDF file 248 to return any extracted value/values to the calling SQL program 202. After generating all the necessary code in the UDF file 248 to set values in an input object, invoke the workflow with the set input object, and extract any return values to return to the calling SQL program 202, the UDF source file 248 is returned (at block 324). The function generator 244 further generates an executable script file 250 that includes statements to register object oriented classes with the DBMS 204 that enable the SQL program 202 to call the UDFs 208a, 208b . . . 208n (paragraph 0144). The above information shows that a second function returns data type information but not data type descriptor;

“registering the one or more data types with a data processing system” as registering data types with a data processing system (paragraph 0129);

“compiling the statement based on the registered one or more data types” as (paragraphs 134, 136, 0052);

"executing the first function to obtain the result data" executing the UDF to obtain the result data (paragraphs 0145, 0117-0118);

"storing the result data obtained from the source" as storing the result data obtained from the resource (paragraphs 0058, 0120);

"returning the result data as data in the format of the registered the one or more data types" as the UDF 208a, 208b . . . 208n statements are executed to set (at block 352) values within the object(s) to pass to the invoked workflow 220a, 220b . . . 220n that include parameters passed with the call from the SQL program 202. The executed statements within the UDF 208a, 208b . . . 208n would then invoke (at block 354) the workflow 220a, 220b . . . 220n by transmitting a workflow function call to the workflow engine 210 with the input object including values set from parameters received from the SQL program 202. Upon receiving (at block 356) a response from the workflow, the UDF 208a, 208b . . . 208n would process (at block 358) the returned output to extract any value(s) returned from the workflow and return (at block 360) extracted value(s) or other information to the SQL program 202 that called the UDF 208a, 208b . . . 208n (paragraph 0145);

For example, the UDF may then be invoked within standard SQL statements such as the following: values getRate(`UK`,`USA`) select item, (price * getRate(country, `USA`)) as cost from purchase_orders (paragraph 0117);
In addition to creating scalar functions such as above, one may also create table functions where the invocation of the web service yields a table of one or more rows,

as represented by the backward arrow in step b) of FIG. 9. For instance, assume that there is a web service that returns status about commercial flights. Using the UDF generating tool one may generate a table function getFlightInfo that returns a table containing the airline, flight number, location, speed, altitude and equipment (paragraph 0118);

"wherein the method is performed by one or more computing devices" as (abstract, paragraph 0031).

Crisan does not explicitly teach the claimed limitation "data type descriptor; in a format that reflects registered one or more data types".

Muthy teaches when an XML document is stored in a database server that supports the XML schema registration techniques described herein, the database server is able to validate the documents to verify that they confirm to the corresponding XML schema. The validation may include validation of both the structure and the datatypes used by the XML document (paragraph 0215).

Murthy teaches creation of SQL Object Types when an XML schema is registered, database server 104 creates the appropriate SQL object types that enable a structured storage of XML documents conforming to this schema. All SQL object types are created in the current user's schema (by default). For example, when PO.xsd is registered, the following SQL types are created. Create type Item_t as object (part varchar2(1000), price number); create type Item_varray_t as varray(1000) of OBJ_T1; create type PurchaseOrder_t as object (purchasedate

date, ponum number, company varchar2(100), item Item_varray_t) (paragraph [0100]).

For example, if the XML schema specifies the datatype to be "string" and a maxLength value of less than 4000, it gets mapped to a varchar2 attribute of the specified length. However, if the maxLength value is not specified in the XML schema, it can only be mapped to a LOB. This is sub-optimal in cases when the majority of string values are actually small--and a very small fraction of them is large enough to necessitate a LOB. The ideal SQL datatype would be varchar2(*) that would perform like varchars for small strings but can accommodate larger strings as well. Further, such columns should support all varchar functionality such as indexing, SQL functions, etc. A similar case can be made for needing a raw(*) datatype to hold unbounded binary values without loss of performance and/or functionality for the small cases (paragraph 0123).

The user can specify SQLType to be NVARCHAR2 for a particular string element or attribute. This ensures that NVARCHAR2 is chosen as the SQL type for the particular element/attribute (paragraph 0126).

Assuming that the XML schema identified by "http://www.oracle.com/P-O.xsd" has already been registered. A XMLType table can be created to store instances conforming to the PurchaseOrder element of this schema--in an object-relational format--as follows: create table MyPOs of xmltype; element "http://www.oracle.com/PO.xsd#PurchaseOder".

Hidden columns are created corresponding to the object type to which the

PurchaseOrder element has been mapped. In addition, a XMLExtra object column is created to store the top-level instance data such as namespaces declarations, etc. Note: XMLDATA is a pseudo-attribute of XMLType that allows directly accessing the underlying object column (abstract, paragraphs 0128-0130).

The above information shows that XML documents are stored based on type information.

Anderson teaches a function returns a char variable as type descriptor containing the full pathname of BLOBs that are stored in external servers to identify a BLOB as data type (col. 26, lines 35-67).

It would have been obvious to a person of an ordinary skill in the art at the time the invention was made to apply Murthy's teaching of storing XML documents based on type information and Anderson's teaching of a function returns a char variable containing the full pathname of BLOBs that are stored in external servers to identify a BLOB as data type to Crisan's system in order to determine how to store subsequently result XML data that conform to the registered XML schema so that the performance and scalability features of the database can be fully exploited to access the result XML data.

As to claims 4, 26, 35, Crisan teaches the claimed limitation "determining that the source is associated with the request by determining whether a certain keyword is specified as a data return type for the first function" as (paragraphs 0142, 0144).

As to claims 5, 27, 36, Crisan teaches the claimed limitation “comprising determining that the source is associated with the request by determining whether the first function returns data in an array of data elements” as (paragraphs 0144, 0067).

As to claims 9, 28, 37, Crisan teaches the claimed limitation “indicates an arrangement of rows and columns of a database table; comprises tabulating the result data according to the arrangement of rows and columns” (paragraph 0067, fig. 15).

Crisan does not explicitly teach the claimed limitation “wherein the data type descriptor, wherein storing the result data obtained from the source in a format that reflects the registered one or more data types”.

Muthy teaches when an XML document is stored in a database server that supports the XML schema registration techniques described herein, the database server is able to validate the documents to verify that they confirm to the corresponding XML schema. The validation may include validation of both the structure and the datatypes used by the XML document (paragraph 0215).

For example, if the XML schema specifies the datatype to be "string" and a maxLength value of less than 4000, it gets mapped to a varchar2 attribute of the specified length. However, if the maxLength value is not specified in the XML schema, it can only be mapped to a LOB. This is sub-optimal in cases when the majority of string values are actually small--and a very small fraction of them is large enough to necessitate a LOB. The ideal SQL datatype would be varchar2(*) that would perform like varchars for small strings but can

accommodate larger strings as well. Further, such columns should support all varchar functionality such as indexing, SQL functions, etc. A similar case can be made for needing a raw(*) datatype to hold unbounded binary values without loss of performance and/or functionality for the small cases (paragraph 0123).

The user can specify SQLType to be NVARCHAR2 for a particular string element or attribute. This ensures that NVARCHAR2 is chosen as the SQL type for the particular element/attribute (paragraph 0126).

Assuming that the XML schema identified by "http://www.oracle.com/P-O.xsd" has already been registered. A XMLType table can be created to store instances conforming to the PurchaseOrder element of this schema--in an object-relational format--as follows: create table MyPOs of xmltype; element "http://www.oracle.com/PO.xsd#PurchaseOder".

Hidden columns are created corresponding to the object type to which the PurchaseOrder element has been mapped. In addition, a XMLExtra object column is created to store the top-level instance data such as namespaces declarations, etc. Note: XMLDATA is a pseudo-attribute of XMLType that allows directly accessing the underlying object column (abstract, paragraphs 0128-0130).

The above information shows that XML documents are stored based on type information.

Anderson teaches a function returns a char variable containing the full pathname of BLOBs that are stored in external servers to identify a BLOB as data type (col. 26, lines 35-67).

It would have been obvious to a person of an ordinary skill in the art at the time the invention was made to apply Murthy's teaching of storing XML documents based on type information and Anderson's teaching of a function returns a char variable containing the full pathname of BLOBs that are stored in external servers to identify a BLOB as data type to Crisan's system in order to determine how to store subsequently result XML data that conform to the registered XML schema so that the performance and scalability features of the database can be fully exploited to access the result XML data.

As to claim 16, Crisan, Murthy and Anderson disclose all of the claimed limitations of claim 16 as discussed in claim 1, except Crisan further teach the claimed limitations "a processing entity; a memory coupled to the processing entity and having program code stored therein which, when executed by the processing entity, causes the processing entity to" as (paragraphs 0147, 0155),

"receive a request to execute the statement that requires computation of the first function included in the program code to return data from a source" as receiving a quest to execute a UDF that included in a program to return data from a source (paragraphs 0145, 0058-0060).

As to claim 22, Crisan, Murthy and Anderson disclose all of the claimed limitations of claim 22 as discussed in claim 1, except Crisan further teach the claimed

limitations “a computer-readable storage medium carrying one or more sequences of instructions which, when executed by one or more processors, causes the one or more processor to” (paragraphs 0147, 0155).

7. Claims 25, 30 and 38 are rejected under 35 U.S.C. 103(a) as being unpatentable over Crisan et al (or hereinafter "Crisan") (US 20030191769) in view of Murthy et al (or hereinafter “Murthy”) (US 20030140308) and Anderson et al (or hereinafter “Anderson”) (US 6047291) and further in view of the admitted prior art of the application.

As to claims 25, 30 and 38, Crisan does not explicitly teach the claimed limitation "wherein the registered one or more data types is used to type-check first function".

The admitted prior art of application teaches the DBMS prcesses the query including type-checking the query using the return type declared for the table function (fig. 2).

It would have been obvious to a person of an ordinary skill in the art at the time the invention was made to apply the admitted prior art of application teaches the DBMS processes the query including type-checking the query using the return type declared for the table function to Crisan's system in order to ensure that the data returned by the table function will be in a pre-defined format that can be returned to the user, avoiding type inconsistencies and errors in data processing system.

Contact Information

8. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Cam Y T. Truong whose telephone number is (571) 272-4042. The examiner can normally be reached on Monday to Firday.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tony Mahmoudi can be reached on (571) 272-4078. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/Cam Y Truong/

Primary Examiner, Art Unit 2169